Building Linux* Kernel with Intel® C++ Compiler for Linux 10.0

White Paper

Feilong Huang

Developer Products Division

Table of content

Introduction

Intel® C++ Compilers have been in the market for over 10 years. More and more software developers are interested in using it to extract better performance on Windows* and Linux*.

As the most essential part of a Linux operating system, Linux Kernel is highly-optimized by the kernel developers. Additionally there are many GNU* C Language extensions, programming tricks and inline assembly code. This makes it challenging for compilers to optimize the Kernel. Building the Linux kernel with Intel C++ Compiler (`icc`) is an ongoing project at Intel. The goal is to improve `gcc`* source compatibility of the Intel C++ Compiler, and to find opportunities to improve kernel performance.

Intel Corporation and Red Flag* Software Co., Ltd, announced that Red Flag was the first company to use the Intel C++ Compiler for Linux to compile a commercial version of its Linux operating system. Details of this announcement are available at
http://www.intel.com/pressroom/archive/releases/20040803net.htm .

Command Line Compatibility

icc does not recognize some gcc options, such as

- `-fno-unit-at-a-time`
- `-msoft-float`
- `-gstabs`
- `-pipe`
- `-mfixed-range` (partially implemented in `icc`)
- `-mregparam=n` (IA-32, Intel 64 only. Implemented in 10.0)
- `-m32` (IA-32, Intel 64 only)

Most of those options are not critical and can be ignored without affecting the objects generated by `icc`. For those options that change the behavior of Linux kernel, we need to replace them with a corresponding icc option. These types of options include `–mfixed-range`, `–mregparam` etc.

The following options are not recommended for use with the Intel C++ compiler when building the Kernel.

```
-Werror
-nostdinc
```

Intel C++ Compiler is stricter in syntax checking and will report more warnings than the GNU compiler. Therefore, -Werror may cause the compiler stop during compilation.

The required substitute header files are supplied with icc for compatibility and performance. The -nostdinc option inhibits the compiler from using those header files.

For example, we have our own va_arg macro in <icc installation dir>/include/xarg.h. With -nostdinc, icc will use GNU va_arg macro as follows.

```
#define va_arg(v,l) __builtin_va_arg(v,l)
```

Unfortunately Intel C++ Compiler does not support __builtin_va_arg. So Intel C++ Compiler will report an error with -nostdinc.

A simple wrapper script to ignore or replace unrecognized compiler options, and then invoke Intel C++ Compiler, can make the command line to compile Linux kernel with Intel C++ Compiler straight forward. In the example script provided here, environment variables HOSTCC and CC will need to be set to the name of the wrapper script.

```
make menuconfig
make HOSTCC=<name of wrapper> CC=<name of wrapper>
make modules_install
```

Building on IA-64

Intel C++ Compiler supports inline assembly code on IA-32 and Intel 64. IA-64 compilers do not support inline assembly. Instead intrinsics that are C-like functions are recommended. Assembly code on IA-64 needs to be rewritten using corresponding intrinsics. Intel C++ Compiler documentation includes a mapping of assembly instructions to intrinsics. Most of these changes have been checked into Linux kernel source tree.

GCC source Compatibility

Some Linux kernel source issues were observed during the compilation of Linux kernel with Intel C++ Compiler. These defects may have been fixed in the newer Linux kernel already.

- **`volatile` attribute**

Look at the following code snippet from `include/asm-ia64/spinlock.h`

```
# define _raw_spin_lock(x)                    \
do {                                    \
      __u32 *ia64_spinlock_ptr = (__u32 *) (x);
     \
     __u64 ia64_spinlock_val;                \
     …                                  \
     if (unlikely(ia64_spinlock_val)) {           \
          do {                          \
               while (*ia64_spinlock_ptr)    \
                    ia64_barrier();         \
          …                             \
          } while (ia64_spinlock_val);       \
     }                                  \
} while (0)
```

In the above code snippet, `ia64_spinlock_ptr` points to a 32-bit volatile data in memory. Without a "`volative`" keyword, compiler may generate the following asm code (shown in pseudo code) for the while loop, which is legal.

```
       load ia64_spinlock_ptr, register
label: test register
       jump-if-not-zero label
```

Unfortunately, the above code results in a dead lock of Linux kernel because the 32-bit data pointed by `ia64_spinlock_ptr` is not reloaded. GNU compiler occasionally generates the "right" code, which is what kernel developers want.

```
label: load ia64_spinlock_ptr, register
       test register
       jump-if-not-zero label
```

In this case, a "`volatile`" attribute is needed for the variable `ia64_spinlock_ptr`, to make sure other compilers won't fail.

- **`inline` keyword**

The `inline` keyword is just a hint to compilers. Compilers may or may not inline an inline function. Here is an instance about inline keyword.
In some applications `gettimeofday()` is a done very often, for example for time stamping all transactions. It would be nice if it could be implemented with very low overhead.
One way of obtaining a fast `gettimeofday()` is by writing the current time in a fixed place, on a page mapped into the memory of all applications, and updating

this location on each clock interrupt. These applications could then read this fixed location with a single instruction - no system call required.
There might be other data that the kernel could make available in a read-only way to the process, like perhaps the current process ID. A `vsyscall` is a "system" call that avoids crossing the userspace-kernel boundary.
`vsyscall()` and `do_vgettimeofday()` are in a special page, which can be accessed in user mode.
Intel C++ Compiler doesn't inline the function "`sync_core`", which is marked as an inline function in `include/asm-x86_64/processor.h`. Thus, the function is compiled as a separate function in the kernel image. `vsyscall()` calls `do_vgettimeofday()` and `do_vgettimeofday()` calls `sync_core()`. The first two functions are called by user applications while `sync_core()` is a kernel function. This will cause a page fault. The following illustrates the call-graph of these 3 functions.

```
     vsyscall( )
          |
  do_vgettimeofday( )                    user space
          |                     --------------------------
     sync_core( )                        kernel space
```

`gcc` happens to inline `sync_core`, so the problem is concealed in gcc-compiled kernel.

Conclusion

Intel® C++ Compiler is highly compatible with GNU Compiler. We've successfully compiled Linux kernel 2.4.21 and 2.6.9 with Intel C++ Compiler on IA-32, Intel® 64 and IA-64, with a small wrapper script and a limited number of temporary source patches.

Additional Information

Intel® Compilers for Linux*: Compatibility with GNU Compilers

For product and purchase information visit:
www.intel.com/software/products